Point Cloud Alignment: Voxelization and Down Sampling

Fatma Güney, Hasan Can Aslan, and Nursena Köprücü

Department of Computer Engineering, Koç University

25 November 2020

1 Report

We had LIDAR scans from different periods of historical buildings and the data is in the form of large-scale point clouds. We basically compared these point clouds to understand the deformation of the structure. In order to apply our algorithm for this data, we first need to prepare data as we can use - and since point clouds consist of millions of points we need to down sample the data. We decided to do voxelization of our data to represent it in a continuous domain. We had two main approaches for this purpose, i.e. Open3D[1] voxelization methods with `voxel_down_sample_and_trace`, and Open3D[1] `voxel_down_sample` with our tracing method.

1.1 Using Open3D Methods

In our first approach, we did voxelization and down sampling with Open3D[1] methods using `voxel_down_sample_and_trace`. This method was successful to downsample the data and it took center points of a voxel as a reference. However, when we get the voxelized point cloud, we want to take their corresponding indexes in the original point cloud. We tried to track and trace methods of Open3D[1] but it did not satisfy our needs. Hence, the issue with this approach was we could not get the points in a voxel in the original point cloud.

```
# Parameters
N = 1000
voxel_size = 0.02

# Import point cloud
pcd_path = "{test_data_dir}/cloud_bin_0.ply"
pcd = o3d.io.read_point_cloud(f"{test_data_dir}/cloud_bin_0.ply")

# Fit to unit cube
min_bound = pcd.get_min_bound()
max_bound = pcd.get_max_bound()
pcd.scale(1 / np.max(max_bound - pcd.min_bound),
          center=pcd.get_center())

# Voxel down sample and trace
out1, out2, out3 = pcd.voxel_down_sample_and_trace(voxel_size=
    voxel_size, min_bound=min_bound, max_bound=max_bound)
```

Code snippet 1: Voxel down sampling with our test data using `voxel_down_sample_and_trace`.

1.2 Using Custom Data Structure for Voxels

In the second approach, we use `voxel_down_sample` with Open3D[1] methods as we did in the first one. However, since we could not get the points that a voxel contains in the original point cloud, we defined our own Cube class. We were able to access the center points of voxels, so we drew a cube around that center point with a given range of x-y-z coordinates. We basically iterated all points in a for loop – and all the voxels in the outer for loop. So we got the points of each voxel inside a cube object. This approach was satisfied our needs and we got the information that we want; however, the issue with this method was it was too slow since we iterated all points and voxels in the for loops.

```
class Cube(object):
    def __init__(self, xrange, yrange, zrange):
        """
        Builds a cube from x, y and z ranges
        """
        self.xrange = xrange
        self.yrange = yrange
        self.zrange = zrange

    @classmethod
    def from_points(cls, firstcorner, secondcorner):
        """
        Builds a cube from the bounding points
        """
        return cls(*zip(firstcorner, secondcorner))

    @classmethod
    def from_voxel_size(cls, center, voxel_size):
        """
        Builds a cube from the voxel size and center of the voxel
        """
        half_center = voxel_size / 2
        x_range = (center[0]-half_center, center[0]+half_center)
        y_range = (center[1]-half_center, center[1]+half_center)
        z_range = (center[2]-half_center, center[2]+half_center)

        return cls(x_range, y_range, z_range)

    def contains_point(self, p):
        """
        Returns given point is in cube
        """
        return all([self.xrange[0] <= p[0] <= self.xrange[1],
                    self.yrange[0] <= p[1] <= self.yrange[1],
                    self.zrange[0] <= p[2] <= self.zrange[1]])
```

Code snippet 2: Our initial custom voxel-like data structure named `Cube`.

```
pcdarray = np.asarray(pcd.points)
downarray = np.asarray(downpcd.points)
voxels = [Cube.from_voxel_size(center, voxel_size) for center in
    downarray]

for point in pcdarray:
    for voxel in voxels:
        if voxel.contains_point(point):
            voxel.points_inside.append(point)
            break
```

Code snippet 3: Splitting our data into voxels using our `Cube` structure.

1.3 Using Matrix Representation

Finally, we optimized our second approach by using matrix representation. Since we vectorized our calculation it became a very fast and efficient version. We defined an x-y-z range around the points that were downsized and decide whether a voxel consists of that point or not, similarly. Here, some points belonged to many voxels and we could not get the same number of points as the original point cloud. This difference was increased when the voxel size was decreased. It was not a big issue but was a note for future work.

```
class Cube(object):
    def __init__(self, range):
        """
        Builds a cube from x, y and z ranges
        """
        self.range = range

    @classmethod
    def from_voxel_size(cls, center, voxel_size):
        """
        Builds a cube from the voxel size and center of the voxel
        """
        cls.center = center
        half_center = voxel_size / 2
        x_range = (center[0]-half_center, center[0]+half_center)
        y_range = (center[1]-half_center, center[1]+half_center)
        z_range = (center[2]-half_center, center[2]+half_center)
        range = np.array([[x_range[0], x_range[1], y_range[0] ,
    y_range[1] ,z_range[0], z_range[1]]])

        return cls(range)

    def contains_points(self, p):
        """
        Returns given point is in cube
        """
        less = np.repeat(self.range, repeats=[p.shape[0]], axis=0)
    [:, 0::2] < p
        greater = np.repeat(self.range, repeats=[p.shape[0]], axis
    =0)[:, 1::2] > p
        filter = np.logical_and(less.all(axis=1), greater.all(axis
    =1))
        return p[filter]
```

Code snippet 4: `Cube` class matrix implementation.

```
pcdarray = np.asarray(pcd.points)
downarray = np.asarray(downpcd.points)

pcds = []
voxels = [Cube.from_voxel_size(center, voxel_size) for center in
    downarray]

for voxel in voxels:
    pcd_voxel = o3d.geometry.PointCloud()
    pcd_voxel.points = o3d.utility.Vector3dVector(voxel.
    contains_points(pcdarray))
    pcd_voxel.paint_uniform_color(get_random_color())
    pcds.append(pcd_voxel)
```

Code snippet 5: Splitting our data into voxels using our `Cube` structure using matrix implementation.

2 Resources

2.1 Source Code

You can find all source code at:

- **GitHub** [https://github.com/hasancaslan/point-cloud-sampling].

2.2 Documentation

You can find documentation about packages for voxelization and down sampling at:

- **Open3D Documentation** [http://www.open3d.org/docs/0.6.0/index.html]
- `voxel_down_sample` documentation.
- `voxel_down_sample_and_trace` documentation.

2.3 Further Questions

For further questions about the project you may contact **Nursena Köprücü** at [nkoprucu16@ku.edu.tr] or **Hasan Can Aslan** at [haslan16@ku.edu.tr].

References

[1] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.